Modelos adjuntos

Olivier Talagrand
Curso Intensivo en Asimilacion de Datos
Buenos Aires, Argentina

5 Noviembre 2008

Variational Assimilation (*4DVar*) minimizes a scalar *objective function* (aka *cost-function*) which measures the misfit between 'model and data' over time interval $[0, K]$.

E. g.

$$\xi_0 \in S \rightarrow$$

$$J(\xi_0) = (1/2)\,(x_0^b - \xi_0)^T\,[P_0^b]^{-1}\,(x_0^b - \xi_0) + (1/2)\,\Sigma_k [y_k - H_k(\xi_k)]^T\,R_k^{-1}\,[y_k - H_k(\xi_k)]$$

subject to *model equation* $\xi_{k+1} = M_k(\xi_k)\,, \qquad k = 0, \ldots, K\text{-}1$

ξ_0 (the model state at time 0 in the present example) is the *control variable* of the minimization. In meteorological applications, it can reach dimension $n \approx 10^6 - 10^7$.

We will denote the control variable $\xi_0 = u$.

$$J(\xi_0) = (1/2)\,(x_0^b - \xi_0)^T\,[P_0^b]^{-1}\,(x_0^b - \xi_0) + (1/2)\,\Sigma_k [y_k - H_k(\xi_k)]^T\,R_k^{-1}\,[y_k - H_k(\xi_k)]$$

Question. How to numerically perform the minimization ?

'Exploration' of control space, by various methods (*e. g.*, *simulated annealing*) ? Cost is prohibitive in large dimension.

The only way so far seems to be through *iterative descent method*

$$u^{(m+1)} = u^{(m)} + \alpha_m D_m$$

where $u^{(m)}$ is m-th approximation of minimizing control variable, D_m is a *descent direction* in control space, and α_m an appropriate scalar coefficient.

Numerous such descent methods exist (many codes are available on software libraries)

- *Steepest descent* (where descent direction D_m is local gradient of objective function). Usually very inefficient.

- *Conjugate gradient* (particularly appropriate for quadratic functions)

- *Newton method* (very efficient as concerns the number of required iterations, but each iteration is very costly)

- *Quasi-Newton methods*

All these methods determine the descent direction from the local gradient

$$\nabla_u \mathsf{J} \equiv (\partial \mathsf{J}/\partial u_i)$$

of J with respect to u and (except for steepest descent) from additional information, in particular gradients computed at previous iterations.

***Question*.** How to numerically determine the required gradient ?

- Analytical expressions ? Forget it.

- Explicit perturbations, producing finite-difference approximation of partial derivatives $\partial J/\partial u_i$?

$$\partial J/\partial u_i \approx [J(u + \delta u_i) - J(u)\ ] / \delta u_i \qquad\qquad i = 1, \ldots, n$$

That would require as many explicit computations of the objective function J as there are components in u. Practically impossible.

Adjoint Method. Elementary principle

Code

Input variables $u_1, u_2, \ldots, u_n$

$*$

$*$

$*$

$a = b \; x \; c$

$*$

$*$

$*$

$J = x_1^2 + x_2^2 + x_3^2$

Purpose. Determine partial derivatives of J with respect to $u_1, u_2, \ldots, u_n$

Adjoint Method. Elementary principle (2)

Input variables $u_1, u_2, \ldots, u_n$

*

*

*

$a = b \; x \; c$

*

*

*

$J = x_1^2 + x_2^2 + x_3^2$

Last instruction

$\partial J/\partial x_1 = 2x_1, \qquad \partial J/\partial x_2 = 2x_2, \qquad \partial J/\partial x_3 = 2x_3$

And then proceed backwards

Adjoint Method. Elementary principle (3)

Operation $a = b \times c$

Input b, c Output a but also b, c

For clarity, we write

$a = b \times c$
$b' = b$
$c' = c$

$\partial J/\partial a$, $\partial J/\partial b'$, $\partial J/\partial c'$ available. We want to determine $\partial J/\partial b$, $\partial J/\partial c$

Chain rule

$$\partial J/\partial b = (\partial J/\partial a)(\partial a/\partial b) + (\partial J/\partial b')(\partial b'/\partial b) + (\partial J/\partial c')(\partial c'/\partial b)$$
$$c \qquad 1 \qquad 0$$

$$\partial J/\partial b = (\partial J/\partial a)\, c + \partial J/\partial b'$$

Similarly

$$\partial J/\partial c = (\partial J/\partial a)\, b + \partial J/\partial c'$$

Adjoint Method. Elementary principle (4)

Operation count. At most 4 times operation count of direct computation u_1, u_2, …, $u_n \rightarrow J$ (in practice of meteorological models, about 2 times). Ratio fundamentally independent of dimension n of input.

BUT it is necessary to keep in memory (or else to recompute in the course of the reverse computation) all quantities such as b and c in above example (more precisely, all quantities which appear in nonlinear operations in direct computation).

Errico and Vukicevic (NCAR, 1991)

Errico and Vukicevic (NCAR, 1991)

Errico and Vukicevic (NCAR, 1991)

Adjoint Method. Slightly less elementary

Input vector $\boldsymbol{u} = (u_i)$, $\dim \boldsymbol{u} = n$

Numerical process, implemented on computer (*e. g.* integration of numerical model)

$$\boldsymbol{u} \rightarrow \boldsymbol{v} = \boldsymbol{G}(\boldsymbol{u})$$

$\boldsymbol{v} = (v_j)$ is *output vector* , $\dim \boldsymbol{v} = m$

Perturbation $\delta\boldsymbol{u} = (\delta u_i)$ of input. Resulting first-order perturbation on $\boldsymbol{v}$

$$\delta v_j = \Sigma_i \, (\partial v_j / \partial u_i) \; \delta u_i$$

or, in matrix form

$$\delta \boldsymbol{v} \; = \; \boldsymbol{G}' \, \delta \boldsymbol{u}$$

where $\boldsymbol{G}' \equiv (\partial v_j / \partial u_i)$ is local matrix of partial derivatives, or *jacobian matrix*, of $\boldsymbol{G}$.

Adjoint Method. Slightly less elementary (2)

$$\delta \boldsymbol{v} = \boldsymbol{G}' \, \delta \boldsymbol{u} \qquad \text{(D)}$$

Scalar function of output

$$\mathrm{J}(\boldsymbol{v}) = \mathrm{J}[\boldsymbol{G}(\boldsymbol{u})]$$

Gradient $\nabla_u \mathrm{J}$ of J with respect to input $\boldsymbol{u}$?

'Chain rule'

$$\partial \mathrm{J}/\partial u_i = \Sigma_j \, \partial \mathrm{J}/\partial v_j \, (\partial v_j/\partial u_i)$$

or

$$\nabla_u \mathrm{J} = \boldsymbol{G}'^{\mathrm{T}} \nabla_v \mathrm{J} \qquad \text{(A)}$$

Adjoint Method. Slightly less elementary (3)

$\boldsymbol{G}$ is the composition of a number of successive steps

$$\boldsymbol{G} = \boldsymbol{G}_N \circ \ \ldots \circ \ \boldsymbol{G}_2 \circ \ \boldsymbol{G}_1$$

'Chain rule'

$$\boldsymbol{G}' = \boldsymbol{G}_N' \ \ldots \boldsymbol{G}_2' \ \boldsymbol{G}_1'$$

Transpose

$$\boldsymbol{G}'^{\mathrm{T}} = \boldsymbol{G}_1'^{\mathrm{T}} \ \boldsymbol{G}_2'^{\mathrm{T}} \ \ldots \ \boldsymbol{G}_N'^{\mathrm{T}}$$

Transpose, or *adjoint*, computations are performed in reversed order of direct computations.

If $\boldsymbol{G}$ is nonlinear, local jacobian $\boldsymbol{G}'$ depends on local value of input $\boldsymbol{u}$. Any quantity which is an argument of a nonlinear operation in the direct computation will be used gain in the adjoint computation. It must be kept in memory from the direct computation (or else be recomputed again in the course of the adjoint computation).

If everything is kept in memory, total operation count of adjoint computation is at most 4 times operation count of direct computation (in practice about 2).

Adjoint Method. The Case of Variational Assimilation

$$J(\xi_0) = (1/2)\,(x_0^b - \xi_0)^T\,[P_0^b]^{-1}\,(x_0^b - \xi_0) + (1/2)\,\Sigma_k[y_k - H_k(\xi_k)]^T\,R_k^{-1}\,[y_k - H_k(\xi_k)]$$

subject to $\xi_{k+1} = M_k(\xi_k), \qquad k = 0, \ldots, K\text{-}1$

Control variable $\xi_0 = \boldsymbol{u}$

Adjoint equation

$$\lambda_K = H_K{}'^T\,R_K^{-1}\,[H_K(\xi_K) - y_K]$$

$$\lambda_k = M_k{}'^T\lambda_{k+1} + H_k{}'^T\,R_k^{-1}\,[H_k(\xi_k) - y_k] \qquad k = K\text{-}1, \ldots, 1$$

$$\lambda_0 = M_0{}'^T\lambda_1 + H_0{}'^T\,R_0^{-1}\,[H_0(\xi_0) - y_0] + [P_0^b]^{-1}\,(\xi_0 - x_0^b)$$

where $H_K{}'^T$ and $M_k{}'^T$ are transpose jacobian matrices of H_k and M_k respectively

Then

$$\nabla_u J = \lambda_0$$

That is exactly what is done in operational 4DVar.